Breve manual de programación en python

Contexto

Este manual se elabora en el marco de la edición 2013 del programa de la ANII "Acortando Distancias", y como parte de la pasantía "Algoritmia e integración de programación al proceso de resolución de problemas en cursos de matemática de la enseñanza media. Didáctica de la Informática." Este es un complemento de un librillo *"Matemática y Programación"*, que contiene los temas trabajados. Todos los materiales están disponibles en www.fing.edu.uy/~darosa/AC2013 y en http://www.anep.edu.uy/prociencia/ .

Introducción

Este manual está dirigido a profesores de matemática de enseñanza media. Su objetivo es que cuenten con una herramienta de rápido y fácil acceso en el momento de trabajar con problemas como los planteados en el librillo u otros similares, en las clases de ciclo básico y bachillerato.
Los ejemplos y comentarios que aquí se utilizan fueron hechos presuponiendo que se utiliza un sistema Linux - Ubuntu 10.04 – Gnome, que está generalmente instalado en las computadoras entregadas a profesores y alumnos.- en el marco del Plan Ceibal
Trabajaremos con el ejemplo: "*Raíces reales de una ecuación de segundo grado*".
El problema consiste en que dados los coeficientes de una ecuación de segundo grado, se quiere obtener sus raíces reales, o un mensaje "esta función no tiene raíces reales"

Intérprete python y comandos directos

Al abrir el intérprete python aparece una ventana en la que podemos escribir. La utilizaremos para aprender algunos comandos en lenguaje python (instrucciones que la computadora debe ejecutar). Si queremos que la computadora escriba una frase en la pantalla por ejemplo *hola mundo,* entonces le daremos la orden para que lo haga, pero claro que la computadora no entiende español, ni inglés ni ningún otro idioma de los que podemos hablar nosotros, solo entiende cosas como 1000101000100 11101001011101010100 0101 que nosotros no comprendemos. Por eso necesitamos alguien que haga de intérprete, es decir que podamos escribir en un lenguaje "humano" y el intérprete lo traduzca al lenguaje de la computadora (código de máquina) y ésta pueda ejecutar lo pedido. Dicho "lenguaje humano" es un lenguaje de programación, en este caso python[1]. Debemos conocer las palabras que hay en este lenguaje y cuáles son su sintaxis y su semántica, pero antes diremos algunas palabras acerca de por qué utilizamos python. La razón principal es que se encuentra instalado en todas las computadoras que el Plan Ceibal les entrega a los alumnos que cursan enseñanza media. Esto lo convierte en una herramienta al alcance de estudiantes y profesores ayudando a lograr los objetivos de la pasantía, es decir, integrar la algoritmia y la programación a los cursos de matemática. Puede también instalarse el intérprete python en cualquier otra computadora. Además, existe en las últimas versiones de GeoGebra un intérprete python que interactúa con los objetos que se estén usando y permite ejecutar comandos y módulos sobre esos objetos. Esta facilidad es muy reciente aún y no se ha desarrollado del todo y por eso no la hemos incluido en este trabajo. Tampoco existe bibliografía suficiente sobre el tema, pero creemos que en un futuro próximo será de gran utilidad y al haber usado python en esta pasantía podremos aprovecharlo rápidamente. Pasamos ahora a describir cómo trabajar con el intérprete python.

1Existen varios lenguajes de programación (Pascal, C, haskell, Java, etc)

Para comenzar abrimos el intérprete python en nuestra pantalla, vamos al menú principal y solo hecemos click un par de veces en:
Aplicaciones > Programación > Python
Ya tenemos una ventana lista para recibir órdenes en lenguaje python, solamente hay que escribir y pulsar Enter.

```
Python 2.6.5 (r265:79063, Oct  1 2012, 22:07:21)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

escribes aquí

Ahora aprendamos algo del intérprete del lenguaje python. En la columna de la izquierda aparecen los comando que deben escribirse en la pantalla y luego pulsar enter, también aparecerán las respuesta que la computadora escribe en la pantalla, mientras que en la columna de la derecha aclaraciones y comentarios sobre los comandos. El intérprete funciona como un evaluador de expresiones:

en python	en español
`>>> 2 + 2`	¿cuánto es 2+2?
`4`	
`>>>4 == 3`	¿4 es igual a 3?
`False`	

Con el comando `print podemos pedirle al intérprete que escriba lo que queramos:`

`>>> print “hola mundo”`	escriba: hola mundo
`hola mundo`	inmediatamente obedece
	observemos que el texto va entre comillas
`>>> print “el area es ”, 2*5`	Puedes probar varios ejemplos para entender la
`el area es 10`	sintaxis del comando *print* (por ejemplo la coma
	antes de 2 * 5)
	NO UTILICES TILDES

```
>>> print “el volumen: “, 2**3, “cm3”
el volumen: 27 cm3
```

Las variables son muy importantes en matemática y en programación, hay variables de muchos tipos, numéricas, alfanuméricas, listas, boolenas, caracteres, etc... comencemos a darle valor a una variable.

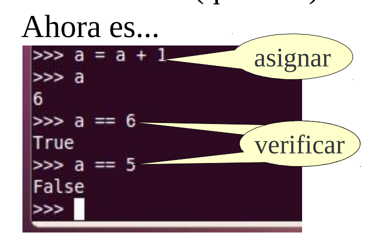
`>>> a = 5`	la variable *a* vale 5, esta línea es una asignación
`>>> a`	para ver que esto es cierto...
`5`	
`>>> a == 4`	¿es a igual a 4?
`False`	no lo es

La igualdad y la asignación

En matemática cuando queremos verificar que dos expresiones denotan el mismo valor, usamos el signo de =. El resultado de la verificación es un valor booleano: verdadero o falso. Por ejemplo, ¿cómo se procede para al evaluar $3 + 1 = 2 + 2$? Se realizan las operaciones para obtener la forma canónica de las expresiones en ambos miembros de la igualdad, y si es la misma, el resultado es verdadero, si no es falso.
Pero el signo de = se usa en matemática también con otro significado: cuando decimos “para $x = 3$ hallar $2x + 1$”, por ejemplo, pretendemos sustituir a la variable *x* por 3 en la expresión y hallar su valor. Esta operación de sustituir a *x* por 3, se denomina asignación y a pesar de que es muy distinta de la verificación de igualdad, en matemática se usa el mismo signo. Esto en programación es inaceptable, dado que la computadora debe saber qué hacer (o *verificar* o *asignar*) y por lo tanto los lenguajes de programación utilizan diferentes símbolos. En python, para la verificación de igualdad se usa el signo = = mientras que para la asignación se usa el =.

```
>>> a = 5
>>> a = a + 1
>>> a
6
>>> a == 6
True
>>> a == 5
False
```

Se asigna 5 a la variable *a*.
El valor de *a* (que es 5) se incrementa en 1.
Ahora es...



Es decir que en la asignación (=) se evalúa la expresión a la derecha del = (con el valor anterior de la variable *a*) y el resultado se asigna a la variable *a* (a la izquierda del signo de =).

El valor de una variable puede ser asignado por el usuario desde el teclado. Para ello se utiliza el comando `input`, veamos un ejemplo.
Deseamos que el usuario pueda indicar el valor de una variable a la que llamaremos *n*.

```
>>> n = input (“ingrese numero mayor que 0: “)
```

Se despliega en la pantalla el mensaje “*ingrese numero mayor que 0:* ”
(Observar que NO ponemos tildes)

```
ingrese numero mayor que 0: _
```

python espera a que el usuario escriba un número y pulse Enter

Una vez que el usuario ingresó un valor, python lo asigna a la variable n.

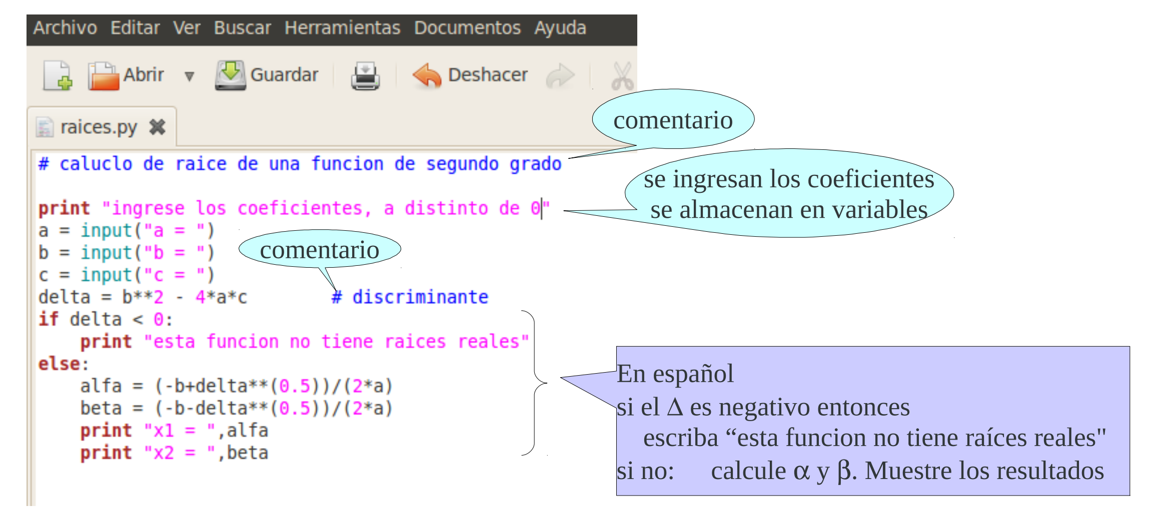
> Resumiendo:
> hay dos formas de asignar valores a las variables,
> o bien directamente en el programa
> o bien ingresados por el usuario.
> Olvidar asignar valores a variables es
> causa frecuente de errores.

Un programa

Para hacer un programa, escribiremos primero todas las instrucciones (comandos) que debe realizar la máquina, en el orden adecuado y luego decimos al intérprete python que las ejecute.

Entonces para empezar debemos utilizar un editor de texto. En esta ocasión es recomendable utilizar gedit, veremos que gedit es capaz de reconocer palabras que nosotros escribimos y destacar con diferentes colores comandos, funciones, números, los comentarios, etc...
Accedemos a él desde el menú principal
Aplicaciones > Accesorios > Editor de textos gedit
El archivo debe ser guardado con extensión .py para que sea reconocido como un programa en lenguaje python, por ejemplo *raices.py*

Podemos hacerlo en cualquier carpeta, pero es importante recordar en cuál fue. En este caso, uno de los siguientes, dependiendo de si es profesor o estudiante:
/home/Profesor/Documentos/Programas *(/home/estudiante/matematica)*
Después de escribir el programa y guardarlo tendremos un documento como el que vemos a continuación. Los colores en las palabras se ven después de haber guardado el archivo con extensión .py
Explicaremos cada paso, pero debes hacerlo utilizando el editor para asegurarte de que estás cuidando cada detalle. Recordemos que son instrucciones para una computadora y cualquier diferencia provocará un mal entendido y la culparemos "injustamente" por no hacer lo que queremos.



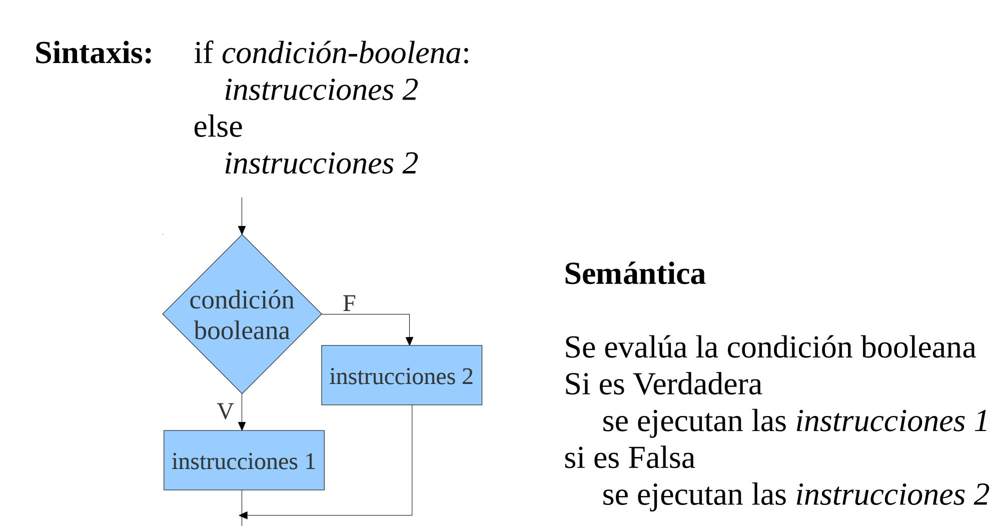
Comentar el programa

El signo # sirve para indicar que empieza un comentario, se puede escribir lo necesario para entender el programa. Puede usarse una línea entera o escribir a la derecha de una instrucción.

El comando if then ... else ...

Para hallar las raíces debemos calcular la raíz cuadrada del discriminante, pero si éste es negativo, no podremos (porque buscamos raíces reales). Los lenguajes de programación proveen *instrucciones de selección* que permiten seleccionar qué instrucciones ejecutar dependiendo de una condición booleana. En python tenemos (observar que then se sustituye por :) :

Sintaxis: if *condición-boolena*:
    instrucciones 2
else
    instrucciones 2



Semántica

Se evalúa la condición booleana
Si es Verdadera
    se ejecutan las *instrucciones 1*
si es Falsa
    se ejecutan las *instrucciones 2*

Veamos cómo funciona el programa

Ahora es el momento de ejecutar el programa que ya está escrito y listo para decirle a la computadora que cumpla con estas instrucciones y el intérprete python será de mucha ayuda.
Primero abriremos una ventana de terminal, esto es fácil de hacer desde el menú principal.
Aplicaciones > Accesorios > Terminal



Una vez que tenemos el terminal vamos al directorio donde habíamos almacenado el archivo.

Escribimos uno de los siguientes dependiendo de si es profesor o estudiante:

```
cd Documentos/Programas     (cd /home/estudiante/matematica)
```

Llamamos al intérprete

```
python
```

Ejecutamos nuestro programa

```
import raices
```

Ahora comienzan a ejecutarse las órdenes una por una ...
Si esto no ocurre hay algún error.
No debes desesperarte.
Nos ha pasado también.

```
>>> import raices
ingrese los coeficientes, a distinto de 0
a = 1
b = 1
c = 1
esta funcion no tiene raices reales
>>> reload (raices)
ingrese los coeficientes, a distinto de 0
a = 1
b = -5
c = 4
x1 =  4.0
x2 =  1.0
<module 'raices' from 'raices.pyc'>
>>> reload (raices)
ingrese los coeficientes, a distinto de 0
a = 1
b = 4
c = 4
x1 =  -2.0
x2 =  -2.0
<module 'raices' from 'raices.pyc'>
>>>
```

El programa pide que ingresemos los coeficientes y devuelve por escrito las raíces llamándolas *x1* y *x2*.
En caso de que la función no tenga raíces escribe en pantalla
"esta funcion no tiene raices reales"
Al finalizar queda el intérprete esperando...

```
>>>_
```

Para volver a ejecutar el programa lo recargamos...

```
>>> reload (raices)
```

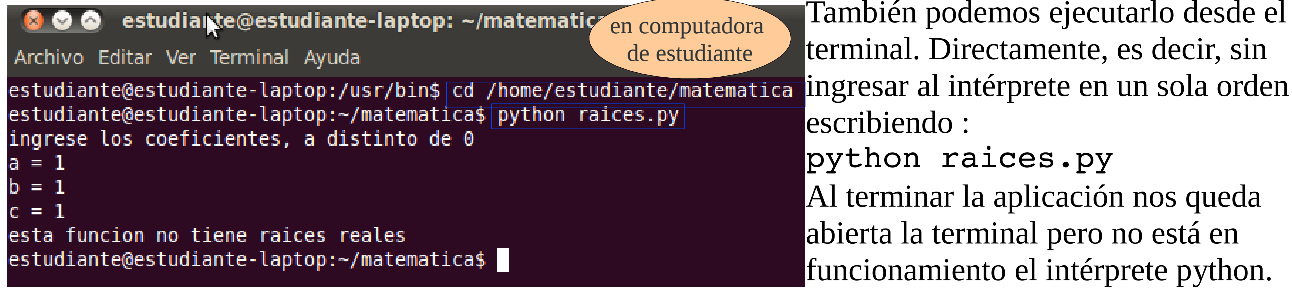
Observemos que si hay una raíz doble, el programa escribe dos veces la misma raíz.

```
x1 = -2.0
x2 = -2.0
```

```
if delta < 0:
    print "esta funcion no tiene raices reales"
else:
    if delta > 0:
        alfa = (-b+delta**(0.5))/(2*a)
        beta = (-b-delta**(0.5))/(2*a)
        print "x1 = ",alfa
        print "x2 = ",beta
    else:    # delta es 0
        raiz_doble = (-b+delta**(0.5))/(2*a)
        print "esta función tiene una raiz dolbe:"
        print "x = ", raiz _doble
```

modificaciones para distinguir el caso de raíz doble

Otro camino



También podemos ejecutarlo desde el terminal. Directamente, es decir, sin ingresar al intérprete en un sola orden escribiendo :

```
python raices.py
```

Al terminar la aplicación nos queda abierta la terminal pero no está en funcionamiento el intérprete python.

Trabajamos con funciones

Un algoritmo puede contener instrucciones englobadas en forma de subalgoritmos para resolver partes del problema. De esta forma el algoritmo (o sea una solución al problema) adquiere una cierta estructura que permite visualizarlo mejor, detectar errores más rápidamente y modificarlo más fácilmente. Asimismo se vuelve más legible. Los subalgoritmos pueden implementarse en python como funciones.

Sintaxis de la definición de función

`def` *nombre_de_funcion* (*entrada*[2]):
   cuerpo
   `return` *salida*

Si en el cuerpo se realizan cálculos sencillos, pueden devolverse directamente en salida (ver definición de *delta* abajo)

```
# coeficientes -> discriminante
def delta(a,b,c):
    return b**2 - 4*a*c

# Cuantas raices tiene?
# coeficientes -> cantidad de raices reales
def cant (a,b,c):
    d = delta (a,b,c)
    if d < 0:
        return 0
    else:
        if d > 0:
            return 2
        else:
            return 1
# Cuales son las raices reales,
# coeficientes -> raices reales
def raices (a,b,c):
    c = cant (a,b,c)
    if c  != 0:
        alfa = (-b + d**(0.5))/(2*a)
        beta = (-b - d**(0.5))/(2*a)
        return (alfa,beta)
```

Uso de otra función

Ejemplo 1:
Función delta, utilizada para calcular el discriminante de una ecuación de segundo grado.
Entrada: Terna de números reales, (coeficientes)
Cuerpo: Cálculo de $b^2 - 4ac$
Salida: Número real, resultado del cálculo anterior (discriminante)

Ejemplo 2:
Función cant, determina la cantidad de raíces reales de una ecuación de segundo grado.
Entrada: Terna de números reales, (coeficientes)
Cuerpo: Calcula delta aplicando la función anterior[3] a los valores *a*, *b* y *c*. Según el valor de *d* (delta(a,b,c)), cant devuelve 0, 1 ó 2.
Salida: Uno de los tres números 0, 1 ó 2.

Ejemplo 3:
Entrada: Terna de números reales, (coeficientes)
Cuerpo: Calcula cant usando la función anterior[3] aplicada a los coeficientes
Salida: Par de números reales (las dos raíces reales)

Grabaremos el archivo de las funciones con el nombre "mod_raices.py" para luego utilizarlo como veremos en la próxima sección.

---

2 Entre paréntesis se escribe el valor de la/las *variable/s*, en programación se le llama *parámetro o argumento.*
3 Observa que las funciones `delta/cant` se encuentran en este mismo archivo (*módulo*).

Cómo utilizar las funciones desde otro módulo

Escribiremos en otro archivo el siguiente módulo en lenguaje python. Éste será un programa que utilizará las funciones del módulo anterior.

```
import mod_raices
print "ingrese los coeficientes (a distinto de 0)"
print "indique S para finalizar o N para seguir"
fin = 'N'
while (fin != 'S'):
    a = input ("a = ")
    if a == 0:
        print "no es ecuacion de 2do. grado"
        fin = raw_input ("finaliza? ")
    else:
        b = input ("b = ")
        c = input ("c = ")
        if mod_raices.cant (a,b,c) == 2:
            rr = mod_raices.raices(a,b,c)
            print "esta funcion tiene dos raices reales"
            print "x1 = ", rr [0]
            print "x2 = ", rr [1]
        else:
            if mod_raices.cant (a,b,c) == 1:
                print "esta funcion tiene una raiz doble"
                print "x = ", mod_raices.raices(a,b,c)[0]
            else:
                print "esta funcion no tiene raices"
        fin = raw_input ("finaliza? ")
print "Gracias por usar mi programa!"
```

`Import mod_raices`
Con este comando le decimos a la computadora que vamos a utilizar las funciones que están definidas en el módulo “mod_raices.py” como se ve en la linea

`mod_raices.cant (a, b, c)`

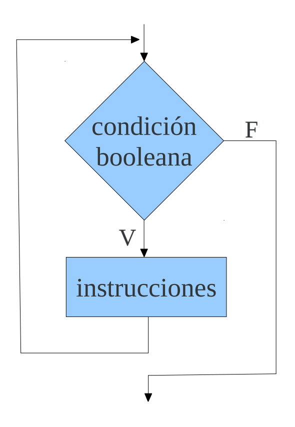
función — módulo — parámetros

rr es un par, *rr* [0] y *rr* [1] son la primera y segunda componente del par respectivamente.

El comando while

Se utiliza para repetir varias veces un mismo conjunto de instrucciones, como se puede ver en el ejemplo anterior.

Sintaxis: `while` *condición-booleana*:
instrucciones



Semántica

1. Se evalúa la condición lógica
   Si es verdadera
      se ejecutan las instrucciones
      se vuelve al paso 1
   Si no, finaliza.

Funciones sobre divisibilidad

Este módulo contiene varias funciones utilizadas en divisibilidad. Podemos escribirlo y guardarlo con el nombre “divis.py” para luego poder utilizar cualquiera de estas funciones.

```
def divisores (a):
    return [x for x in range (1,a+1) if a%x == 0]

def esprimo (a):
    d = divisores (a)
    return d == [1,a]

def inter (l1,l2):
    l = []
    for element in l1:
        if element in l2:
            l.append (element)
    return l

def divisorescom (a,b):
    d1 = divisores (a)
    d2 = divisores (b)
    dc = inter (d1,d2)
    return dc

def mcd (a,b):
    dc = divisorescom (a,b)
    m = max (dc)
    return m

# esta es una funcion recursiva
# utiliza el algoritmo de euclides
def MCDr (a,b):
    r = a%b
    if r == 0:
        return b
    else:
    return MCDr (b,r)
```

Veamos algunos ejemplos en el intérprete python

¿7 es número primo?

```
>>> import divis
>>> divis.esprimo(7)
True
>>>
```

También...

¿Cuál es el MCD entre 6 y 15?

¿Cuáles son los divisores de 12?

¿Cuáles son los divisores comunes de 24 y 20?

```
>>> import divis
>>> divis.mcd(6,15)
3
>>> divis.divisores(12)
[1, 2, 3, 4, 6, 12]
>>> divis.divisorescom(24,20)
[1, 2, 4]
>>>
```

¿Cuáles son los números primos entre 1 y 30?

```
>>> [x for x in range (1, 31) if divis.esprimo(x)]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
>>>
```

Buenas prácticas de programación

Pensar en integrar la programación en los cursos de matemática, nos lleva a decir unas palabras sobre la actividad de programar. Suele suceder que una vez que una persona aprende las nociones básicas de un lenguaje de programación, y comienza a ejecutar programas, se genera una especie de "círculo vicioso" por el cual la persona intenta corregir errores o mejorar su programa directamente en la máquina, y su trabajo creativo de diseño de la solución del problema, queda relegado. Esto tiene varias desventajas, destacaremos dos: por un lado, la corrección de un error suele generar otros en otras partes del programa (que puede consistir de varios módulos), y por otro lado, se produce un desfasaje entre la implementación y el diseño original, y suele ser difícil de reconstruir la solución definitiva. Lo que es deseable y constituye una buena práctica de programación recomendada por los académicos, es que frente a errores, se vuelva al diseño y se corrijan los errores en la solución elaborada. Luego, se corrija la implementación.
Otra recomendación consiste en conocer las herramientas que provee el lenguaje utilizado y usarlas correctamente. Muchas veces, un mismo resultado se logra utilizando distintas herramientas, y en algunos casos, se logra el resultado esperado pero utilizándolas mal.
Ejemplo: sea el problema: dado un número *n*, hallar los múltiplos de *n* menores o iguales que una cierta cota (en este caso la cota es 19**n*). Un programa en python podría ser:

```
n = input("ingrese numero:")     # se ingresa n
i = 0                            # partimos de 0
r = []                           # resultado
while i <= 19:                   # iteramos hasta 19
    m = n*i                      # m es multiplos de n
    r  = r + [m]                 # se agrega al resultado
    i = i + 1                    # nueva iteracion
print r                          # cuando i = 20 mostramos el resultado
```

El exceso de comentarios es para ilustrar, conviene usar menos

Funciona. Sin embargo python provee otras herramientas que permiten hacer un programa más cercano a la definición matemática de múltiplo, más corto, más sencillo y más eficiente:

```
n = input("ingrese numero:")    # se ingresa n
r = [n*x for x in range (20)]   # el resultado es la lista de n*x para 0 <= x < 20
print r
```

Consideremos este segmento de programa:

```
def raices (a,b,c):
  delta = math.sqrt (b**2 - 4*a*c)
    if delta > 0:
        r1 = (-b + delta) / (2*a)
        r2 = (-b - delta) / (2*a)
    return (r1, r2)
```

El mismo tiene un error *en el algoritmo*, que es que denomina delta a *la raíz de* $(b^2 - 4 * a * c)$, por lo tanto si ese valor es negativo el programa devolverá un error. Es recomendable revisar el diseño y corregir el error en él, en vez de intentar corregir directamente en el programa. Muchas veces los errores son más fáciles de encontrar en el diseño del algoritmo que en el programa, donde muchas veces (la mayoría) una corrección puede generar otros errores y se vuelve difícil la corrección.

Posibles errores

Es posible que cometas algunos errores en el momento de implementar en lenguaje python tus programas y funciones, por eso te contamos aquí cuáles fueron las cosas que hicimos mal y cómo el intérprete nos lo *dijo*.

```
SyntaxError: invalid syntax
ImportError: No module named raices
SyntaxError: invalid syntax
SyntaxError: EOL while scanning string literal
```

Las letras no salen en color cuando estoy escribiendo en el gedit”

Debes guardar el archivo con extensión .py

No poner tildes.

```
Ejemplo:
print “raíces”
```

```
SyntaxError: Non-ASCII character
```

```
Print “raíces”
```

Me olvidé la indentación, es decir no hice la tabulación de forma adecuada
Ejemplo:

```
while i > 1:
m = m * i
i = i +1
```

```
IndentationError: unexpected indent
```

las instrucciones están en el mismo nivel de indentación que while

Me olvidé los dos puntos
Ejemplo:

```
if a > 0
     m = m * n
```

```
SyntaxError: invalid syntax
```



Me olvidé de los paréntesis en reload

```
SyntaxError: invalid syntax
```

Puse el archivo en otra carpeta

```
>>> import raices
ImportError: No module named raices
```

Olvidé cerrar las comillas

```
Ejemplo:
print “ingrese numero
```

```
SyntaxError: EOL while scanning
string literal
```

```
Print “ingrese un numero”
```

Operaciones aritméticas

- Resta: -
- Suma: +
- Multiplicación: *
- División entera:

  cociente: //

  resto (módulo): %

  (2 // 3 es 0, 2 % 3 es 2)
- División (para reales): / (2.0 / 3.0 es 0.66666)
- Exponente: ** (a ** b es a^b)

Expresiones booleanas

Constantes: True, False

Operadores:

- `and` - conjunción
- `or` - disyunción
- `not` – negación

Operadores relacionales

= = (igual)

! = (distinto)

< = (≤, menor o igual)

> = (≥, mayor o igual)

> (mayor)

< (menor)

Para información sobre funciones predefinidas en python ver http://mundogeek.net/tutorial-python/